Karlsruhe Institute of Technology

HCI ACCESSIBILITY

Praxis der Softwareentwicklung

Solidarische Raumnutzung

Praxis der Softwareentwicklung (PSE)
Wintersemester 2024/25

Testbericht

Karlsruher Institut für Technologie (KIT)
Institut für Anthropomatik und Robotik (IAR)
Forschungsgruppe Mensch-Maschine-Interaktion und Barrierefreiheit (MBI)
Prof. Dr. Kathrin Gerling
Adenauerring 10, Gebäude 50.28
76131 Karlsruhe

Betreuer: Sabrina Burtscher, Dmitry Alexandrovsky

Projektteilnehmer:

Name	E-Mail-Adresse
Antonia Ammon	uipkm@student.kit.edu
Ben Steinle	ufikl@student.kit.edu
Johannes Frohnmeyer	uczkf@student.kit.edu
Alexander Klee	ukvpq@student.kit.edu
Jannik Hönlinger	uouyo@student.kit.edu

Karlsruhe, 26. März 2025

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Ziel dieses Projekts ist es, eine Webanwendung *Soli* zur solidarischen Raumnutzung zu entwickeln. Die Website soll das Buchen von Räumen ermöglichen, wobei der Fokus darauf liegt, Buchungen mit differenzierten Prioritäten zu ermöglichen.

Gegenstand dieses Testberichts ist die Qualitätssicherung der Anwendung. Dabei liegt der Fokus auf den durchgeführten Tests, den Testergebnissen und der Bewertung der Systemqualität. Zusätzlich werden die verwendeten Testmethoden und -werkzeuge erläutert.

2 Szenarien

Nr.	Testfälle Beschreibung
⟨T10⟩	Landeseite besuchen
⟨T20⟩	Login
⟨T30⟩	Abmelden
⟨T40⟩	Reservieren
⟨T50⟩	Löschen von Terminen durch Administratoren
⟨T60⟩	Deaktivieren eines Kontos
⟨T70⟩	Benachrichtigung bei freiem Raum
⟨T80⟩	Anzeige des Raumstatus
⟨T90⟩	Stornierung einer Reservierung
⟨T100⟩	Login mit Adminkonto
⟨T110⟩	Deaktivierung von Gastkonten
⟨T120⟩	Öffnungszeiten einstellen
⟨T130⟩	Terminkonfliktauflösung

Tabelle 2.1: Überblick der Testfälle.

Nr.	Testszenarien Beschreibung	Ergebnis
⟨S10⟩	Besuch der Landeseite und Anmeldung/Abmeldung	Positiv
⟨S20⟩	Reservierung eines Raums und Stornierung	Positiv
⟨S30⟩	Terminverwaltung durch das Adminkonto	Positiv
⟨S40⟩	Kontoverwaltung durch das Adminkonto	Positiv
⟨S50⟩	Terminkonflikt auflösung	Positiv

Tabelle 2.2: Überblick der Testszenarien.

2.1 Testfallszenarien

Szenario: Besuch der Landeseite und Anmeldung/Abmeldung⟨$S10$⟩

Ziel: Sicherstellen, dass Nutzende die Landeseite aufrufen und sich erfolgreich anmelden bzw. abmelden können.

1. Der Nutzende besucht die Landeseite ⟨T10⟩.
2. Der Nutzende loggt sich ein ⟨T20⟩.
3. Der Nutzende meldet sich ab ⟨T30⟩.

Ergebnis: Positiv. Die An- und Abmeldung funktioniert für KIT Konten, Gastkonten sowie Admin Konten.

Szenario: Reservierung eines Raums und Stornierung⟨$S20$⟩

Ziel: Überprüfen, ob Nutzende erfolgreich Termine reservieren können.

1. Der Nutzende besucht die Landeseite ⟨T10⟩.
2. Der Nutzende meldet sich an ⟨T20⟩.
3. Der Nutzende wählt einen verfügbaren Raum aus und reserviert diesen ⟨T40⟩.
4. Buchen ausserhalb der Öffnungszeiten sollte hierbei fehlschlagen.
5. Überprüfen, ob der Raumstatus korrekt angezeigt wird und sich ggf. geändert hat ⟨T80⟩.
6. Der Nutzende storniert die Reservierung ⟨T90⟩.
7. Andere Nutzende, welchen sich diesen Termin vorgemerkt hatten, werden benachrichtigt, dass der Raum nun Frei ist ⟨T70⟩.

Ergebnis: Positiv. Die Reservierung von Räumen funktioniert, sowie die Stornierung von Reservierungen. Dabei werden Nutzende darauf hingewiesen, dass sie den Raum nicht buchen können, wenn sich die Buchung außerhalb der Öffnungszeiten befindet.

Szenario: Terminverwaltung durch das Adminkonto⟨$S30$⟩

Ziel: Testen der administrativen Funktionalitäten zum Löschen von Terminen.

1. Ein Admin besucht die Landeseite ⟨T10⟩.

2. Der Admin meldet sich an ⟨T100⟩.
3. Der Admin löscht bestehende Termine ⟨T50⟩.
4. Der Admin meldet sich ab ⟨T30⟩.

Ergebnis: Positiv. Admins können sich anmelden und genauso wie reguläre Konten Termine erstellen als auch löschen.

Szenario: Kontoverwaltung durch das Adminkonto⟨$S40$⟩

Ziel: Testen der administrativen Funktionalitäten zum Deaktivieren von Gastkonten und einstellen der Öffnungszeiten.

1. Ein Admin besucht die Landeseite ⟨T10⟩.
2. Der Admin meldet sich an ⟨T100⟩.
3. Der Admin deaktiviert die Anmeldung von Gastkonten ⟨T110⟩.
4. Der Admin ändert die Öffnungszeiten für einen bestimmten Wochentag ⟨T120⟩.
5. Der Admin öffnet die Ansicht *Kontoliste* und deaktiviert ein Konto ⟨T60⟩.
6. Der Admin meldet sich ab ⟨T30⟩.
7. Das Anmelden mit einem Gastkonto sollte jetzt fehlschlagen ⟨T20⟩.
8. Das Anmelden mit dem deaktiviertem Konto sollte ebenfalls fehlschlagen ⟨T20⟩.
9. Die Ansicht *Kalender* sollte die neuen Öffnungszeiten darstellen.

Ergebnis: Positiv. Admins können Konten deaktivieren und die Öffnungszeiten ändern, sowie die Gastkonto funktionalität insgesamt ausschalten.

Szenario: Terminkonflikt auflösung⟨$S50$⟩

Ziel: Überprüfen, ob ein Terminkonflikt richtig aufgelöst wird.

1. Konto 1 meldet sich an ⟨T20⟩.
2. Konto 1 erstellt einen Termin, und gibt die zu testende Priorität, Raumteilungsoption und Zeitperiode ein ⟨T20⟩.
3. Konto 1 meldet sich ab und Konto 2 meldet sich an.
4. Konto 2 erstellt einen Termin, welcher den anderen überlappt.
5. Es wird überprüft, ob die erwartete Konfliktauflösung stattgefunden hat und ggf. die dafür benötigten E-Mails versendet wurden ⟨T130⟩.

Ergebnis: Positiv. Terminkonflikte werden erkannt und die Nutzenden per E-Mail benachrichtigt.

3 Werkzeuge

In diesem Kapitel werden die verschiedenen Werkzeuge beschrieben, die im Projekt verwendet wurden, um die Entwicklung, das Testen und die Qualitätssicherung zu unterstützen.

3.1 JUnit

JUnit ist ein Framework zum Schreiben und Ausführen von Unit-Tests in Java. Es ermöglicht das Testen einzelner Methoden und Klassen, um sicherzustellen, dass sie wie erwartet funktionieren. Wir setzen dieses Framework für Unittests auf allen Ebenen sowie für Integrationstests ein. Damit können wir sicherstellen, dass der Code korrekt funktioniert und Fehler frühzeitig identifizieren. Da bereits in der Implementierungsphase Unittests für die einzelnen Komponenten geschrieben wurden, konnten wir sicherstellen, dass unsere Anpassungen in der Qualitätssicherung keine neuen Fehler eingeführt haben. Außerdem haben wir weitere Tests geschrieben, um sicherzustellen, dass die neuen Funktionen korrekt funktionieren.

3.2 JaCoCo

JaCoCo (Java Code Coverage) ist ein Werkzeug zur Messung der Testabdeckung. Es zeigt an, welche Teile des Codes durch Tests abgedeckt sind und welche nicht, was hilft, ungetesteten Code zu identifizieren. Wir haben JaCoCo verwendet, um sicherzustellen, dass unsere Tests eine ausreichende Abdeckung des Codes haben. Dadurch können wir Fehler durch ungetesteten Code vermeiden und die Qualität des Codes verbessern.

Zum Ende der Implementationsphase hatten wir mit mehr als 150 Tests eine Coverage von 67% erreicht. In der Qualitätssicherung haben wir die Coverage auf 76% erhöht. Eine detaillierte Aufschlüsselung der Coverage in den verschiedenen Paketen finden Sie in Tabelle 3.1.

Paket	Line Coverage
Controller	69% -> 77%
Domain	87% -> 91%
DTO	79% -> 83%
Filter	100% -> 90%
Repository	100%
Service	85% -> 84%
Gesamt	67% -> 76%

Tabelle 3.1: Coverage der verschiedenen Pakete

3.3 GitHub CI

GitHub CI (Continuous Integration) ist ein Dienst, der automatisch Builds und Tests ausführt, wenn Code in ein GitHub-Repository eingecheckt wird. Dies hilft, sicherzustellen, dass der Code immer in einem funktionsfähigen Zustand ist. Wir setzten GitHub CI sowohl bei jedem Commit als auch bei jedem Pull-Request ein, um unsere Tests auszuführen, die Coverage zu überprüfen und Encoding-Probleme zu identifizieren. Dadurch konnten wir den Review-Prozess beschleunigen und sicherstellen, dass der Code immer in einem funktionsfähigen Zustand ist.

3.4 Security Tests

GitHub bietet auch die Möglichkeit, automatische Sicherheitstests und Abhängigkeitsanalysen durchzuführen. Mit diesen Tests können Sicherheitslücken und Schwachstellen in den Abhängigkeiten des Projekts identifiziert werden. Außerdem nutzen wir Dependabot, um automatisch Abhängigkeiten zu aktualisieren und Sicherheitslücken zu schließen.

3.5 Codequalität

Zur Überprüfung und Verbesserung der Codequalität haben wir die Analysewerkzeuge von IntelliJ IDEA verwendet. Diese Werkzeuge helfen, potenzielle Probleme im Code zu identifizieren und zu beheben, um die Qualität des Codes zu verbessern. In Kombination mit ausführlichen Code-Reviews konnten wir sicherstellen, dass der Code so sauber und wartbar wie möglich ist. Weitere Werkzeuge wie Checkstyle, Google Java Format, Palantir Java Format, Error Prone und andere wurden als Optionen zur Automatisierung dieses Prozesses in Betracht gezogen und evaluiert, zum Vermeiden des durch sie verursachten Mehraufwands durch aufwändige Konfiguration für bestenfalls minimale Verbesserungen aber nicht eingesetzt.

4 Bugs

Im Verlauf der Qualitätssicherung wurden zahlreiche Fehler gefunden und behoben. Genauer wurden insgesamt 103 Issues behoben. (Issues sind hierbei Fehler, die auf GitHub als Issues gemeldet wurden. Da nicht alle Issues tatsächliche Fehler sind, werden hier nicht alle Issues als Fehler betrachtet.) Von diesen waren vor der Qualitätssicherung bereits 68 Fehler behoben, welche hier nicht aufgeführt sind. Die folgenden Abschnitte beschreiben die gefundenen Fehler und die durchgeführten Korrekturen.

4.1 NPE beim Erstellen eines Raums ohne Beschreibung

Fehlersymptom: Beim Erstellen eines Raums ohne Beschreibung tritt ein NullPointerException (NPE) auf.
Fehlergrund: Die Beschreibung des Raums wird nicht auf null überprüft, bevor sie verwendet wird.
Behebung: Eine Überprüfung auf null bevor die Beschreibung verwendet wird wurde hinzugefügt. Im Fall eines Fehlers wird eine alternative Zeichenkette als Standardwert gesetzt.

4.2 Correct Z-Order for drawer on mobile

Fehlersymptom: Der Drawer auf mobilen Geräten wird unter dem Kalender angezeigt.
Fehlergrund: Die Z-Reihenfolge der Elemente wurde nicht korrekt festgelegt.
Behebung: Die Z-Reihenfolge der Elemente wurde korrekt festgelegt, sodass der Drawer über dem Kalender liegt.

4.3 Use Hamburger menu on mobile

Fehlersymptom: Der Drawer auf mobilen Geräten wird nicht durch ein Hamburger-Menü geöffnet.

Fehlergrund: Das Hamburger-Menü wurde nicht implementiert.
Behebung: Ein Hamburger-Menü wurde hinzugefügt, um den Drawer auf mobilen Geräten zu öffnen.

4.4 New icon for ShareRoomType.REQUEST

Fehlersymptom: Das Icon für den Raumtyp ShareRoomType.REQUEST ist nicht eindeutig.
Fehlergrund: Das Icon für ShareRoomType.REQUEST wurde schlecht gewählt.
Behebung: Ein neues Icon für ShareRoomType.REQUEST, das den Raumtyp besser repräsentiert, wurde hinzugefügt.

4.5 book at least 15 minutes into the future

Fehlersymptom: Es ist möglich, einen Raum in der Vergangenheit zu buchen.
Fehlergrund: Es wurde keine Überprüfung hinzugefügt, um sicherzustellen, dass der Raum mindestens 15 Minuten in der Zukunft gebucht wird.
Behebung: Eine Überprüfung um sicherzustellen, dass der Raum mindestens 15 Minuten in der Zukunft gebucht wird, wurde hinzugefügt.

4.6 Email benachrichtigung fehlen bei Akzeptierung oder Löschung durch Deaktivierung der Gastfunktion

Fehlersymptom: Es werden keine E-Mails versendet, wenn ein Raum akzeptiert oder gelöscht wird, weil die Gastfunktion deaktiviert wurde.
Fehlergrund: Das Versenden von E-Mails in diesen Fällen wurde nicht implementiert.
Behebung: Das Versenden von E-Mails in diesen Fällen wurde implementiert.

4.7 FullCalendar ist auf kleinen Bildschirmen schwer zu bedienen

Fehlersymptom: Der FullCalendar ist auf kleinen Bildschirmen schwer zu bedienen, da die Schaltflächen zu klein sind.

Fehlergrund: Die Monatsansicht wurde als Standardansicht festgelegt, was auf kleinen Bildschirmen nicht gut funktioniert.
Behebung: Die Tagesansicht wurde bei kleinen Bildschirmen als Standardansicht gestgelegt, um die Bedienung auf kleinen Bildschirmen zu erleichtern

4.8 Die Konfliktlösungs-UI nutzt rohen Text

Fehlersymptom: Die Konfliktlösungs-UI zeigt rohen Text an, anstatt eine benutzerfreundliche Oberfläche zu verwenden.
Fehlergrund: Eine benutzerfreundliche Oberfläche wurde nicht implementiert.
Behebung: Eine benutzerfreundliche Oberfläche für die Konfliktlösung auf Basis der List-Komponente wurde implementiert

4.9 Buchungen mit offenen Anfragen werden nicht markiert

Fehlersymptom: Buchungen mit offenen Anfragen werden nicht als solche markiert.
Fehlergrund: Die Markierung von Buchungen mit offenen Anfragen wurde nicht implementiert.
Behebung: Die Markierung von Buchungen mit offenen Anfragen wurde implementiert.

4.10 Correct sharing icon position in calendar

Fehlersymptom: Die vertikale Position des Sharing-Icons im Kalender ist nicht korrekt.
Fehlergrund: Die vertikale Position des Sharing-Icons wurde nicht korrekt festgelegt.
Behebung: Die vertikale Position des Sharing-Icons wurde korrekt festgelegt.

4.11 Room still selected after deletion

Fehlersymptom: Ein Raum ist immer noch ausgewählt, nachdem er gelöscht wurde. Dies führt zu Fehlern, wenn versucht wird, mit dem gelöschten Raum zu interagieren.
Fehlergrund: Die Auswahl des Raums wurde nicht aufgehoben, nachdem der Raum gelöscht wurde.

Behebung: Die Auswahl des Raums wird aufgehoben, nachdem der Raum gelöscht wurde.

4.12 Doppelung Aria-Labels zu Tooltips und normaler Text

Fehlersymptom: Der Ihalt der Aria-Labels war gleich zu dem der Tooltips oder sogar dem normalen Text.
Fehlergrund: Die Logik der Textreader wurde falsch verstanden.
Behebung: Die Aria-Labels und Tooltips wurden angepasst, sodass der Inhalt nicht mehr doppelt vorliegt.

4.13 Tooltips für Raumteilungssymbole

Fehlersymptom: Die Terminansicht konnte nicht geladen werden.
Fehlergrund: Die Terminansicht wurde nicht korrekt geladen, da die Tooltips für die Raumteilungssymbole nicht korrekt gesetzt wurden.
Behebung: Die Tooltips für die Raumteilungssymbole wurden korrekt gesetzt, sodass die Terminansicht geladen werden kann.

4.14 Buchungsende wird in UI nicht gegen Start validiert

Fehlersymptom: In der Benutzeroberfläche wird ein Enddatum vor dem Startdatum nicht sofort als Fehler angezeigt.
Fehlergrund: Die Validierung des Enddatums gegen das Startdatum wurde erst nach dem Absenden des Formulars durchgeführt.
Behebung: Die Validierung des Enddatums gegen das Startdatum wird sofort durchgeführt.

4.15 Bannerzustand wird nicht korrekt aktualisiert

Fehlersymptom: Der Zustand des Banners wird nicht korrekt aktualisiert, wenn der Raum gewechselt wird.
Fehlergrund: Die Berechnung des neuen Zustandes nutzte den alten Raum.

Behebung: Die Berechnung des neuen Zustandes nutzt den neuen Raum.

4.16 Round bookings

Fehlersymptom: Buchungen, die nicht im Client gerundet wurden, führen zu einer Fehlermeldung.
Fehlergrund: Buchungen, die nicht im Client gerundet wurden, führen zu einer Fehlermeldung.
Behebung: Buchungen werden zusätzlich im Server gerundet, um Fehlermeldungen zu vermeiden.

4.17 Korrekte darstellung der Priorität-badges

Fehlersymptom: Die Priorität-Badges werden nicht korrekt dargestellt und nicht übersetzt.
Fehlergrund: Die Text-attribute der Badge wurde überschrieben.
Behebung: Die Text-attribute der Badge wird nicht mehr überschrieben.

4.18 Pagination: *Seite X von Y* hat hover effekt

Fehlersymptom: Bei einer Gliederung einer Liste auf mehrere Seiten wird der Text *Seite X von Y* als Button dargestellt und wird somit hervorgehoben wenn die Maus sich darüber befindet.
Fehlergrund: Falsche CSS Klassen wurden verwendet.
Behebung: Füge richtige Css Klassen hinzu.

4.19 Footer überdeckt Sidebar auf kleineren Bildschirmen

Fehlersymptom: Der Footer überdeckt die Sidebar auf kleineren Bildschirmen.
Fehlergrund: Der Footer wurde fix positioniert, was zur Folge hatte, dass seine Höhe nicht berücksichtigt wurde.
Behebung: Die Sidebar wurde mittels Flexbox so angepasst, dass der Footer keine besondere Positionierung benötigt.

4.20 Fehlerdialoge sind nicht zentriert

Fehlersymptom: Fehlerdialoge sind nicht zentriert.
Fehlergrund: Die Positionierung der Fehlerdialoge wurde nicht korrekt festgelegt.
Behebung: Die Positionierung der Fehlerdialoge wurde korrekt festgelegt.

Glossar

CSS Cascading Style Sheets, ist eine Sprache um das Visuelle aussehen einer Website zu definieren. 15